Objekt-basiert oder objekt-orientiert?
Moderne Low Level Treiberprogrammierung mit C/C++

Renate Schultes
MicroConsult GmbH

MICROCONSULT

Agenda

- Programmier-Paradigmen
- Programmiersprachen C und C++
- Aufgabenstellung "Zähler"
  - Lösung Objekt-basiert (C)
  - Lösung Objekt-orientiert (C++)
- Low-Level Treiber in C und C++
- Zusammenfassung

MICROCONSULT

Agenda

- **Programmier-Paradigmen**
- Programmiersprachen C und C++
- Aufgabenstellung "Zähler"
  - Lösung Objekt-basiert (C)
  - Lösung Objekt-orientiert (C++)
- Low-Level Treiber in C und C++
- Zusammenfassung

Prozedurale Programmierung

- Ein Algorithmus wird in überschaubare Teile zerlegt.
  ➪ Dafür werden Prozeduren und Funktionen verwendet.
- Daten werden global bereitgestellt
  ➪ uneingeschränkter Zugriff
- Daten werden lokal definiert
  ➪ und bei Bedarf als Parameter übergeben

➪ Funktionen und Daten hängen nicht zusammen

Modularisierung
Das Gesamtprogramm wird in überschaubare Teile (Module) zerlegt.

Zur Steuerung der Befehlsausführung werden Kontrollstrukturen

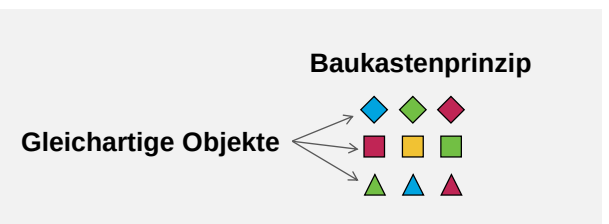
- **Sequenzen**
  Anweisungen, die nacheinander ausgeführt werden
- **Verzweigungen**
  Entscheidungen
- **Schleifen**
  Wiederholungen

eingesetzt.

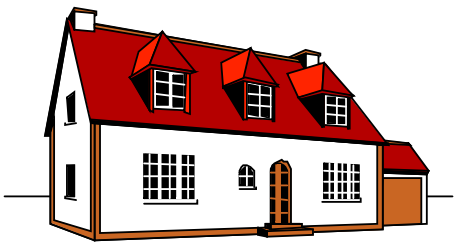
Objekt-basierte Programmierung

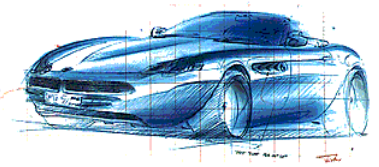
Der Programmfluss basiert auf fertigen Objekten, die zum Beispiel von der Run-Time Umgebung zur Verfügung gestellt werden.
➪ zum Beispiel Microsoft Powershell

Eine Vorstufe der objekt-orientierten Programmierung, in der es keine Vererbung gibt.
➪ Zur einfacheren Verwaltung von gleichartigen Objekten



Was ist ein Objekt?





Objekte können etwas zum Anfassen sein, wie zum Beispiel ein Haus, ein Auto, eine Ampel oder ein Mikrocontroller. Aber auch abstrakte Objekte, wie zum Beispiel Personendaten oder Buchungen.

Reale Objekte, die im Umfeld einer zu lösenden Aufgabe vorkommen, können elektronisch nachgebildet werden.

Objekt-orientierte Programmierung OOP

Die OOP bildet die Aufgabenstellung in Form von Objekten ab.

Grundbausteine sind:

- **Abstraktion** - Daten und Funktionen werden zusammengefasst (in eine Klasse)
- **Kapselung** - Daten werden nach außen hin gekapselt, nicht zum Objekt gehörende Methoden dürfen nicht auf diese zugreifen
- **Vererbung** - abgeleitete Klassen erben Daten und Funktionen der Basisklasse
- **Polymorphie** – eine Methode nimmt abhängig von ihrer Verwendung unterschiedliche Typen an

➪ Ein Objekt ist eine Sammlung von Daten und Funktionen. Der **Zusammenhang zwischen Daten und Funktionalität** ist klar erkennbar.

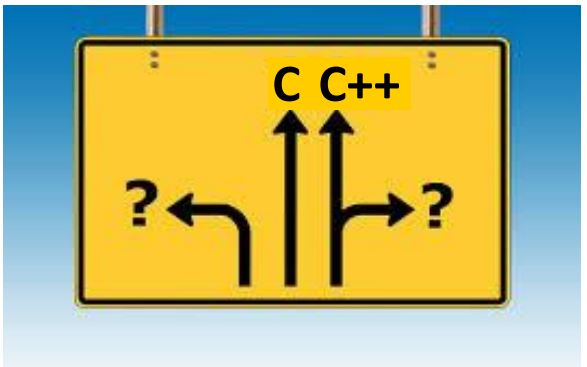
Auszug aus Wikipedia:

Der ISO/IEC 2382-15-Standard von 1999 definiert den Begriff *object-oriented* dagegen wie folgt:
Bezieht sich auf eine Technik oder Programmiersprache, welche Objekte, Klassen und Vererbung unterstützt.“[1]

[1] https://de.wikipedia.org/wiki/Objektorientierte_Programmierung

Agenda

- Programmier-Paradigmen
- **Programmiersprachen C und C++**
- Aufgabenstellung "Zähler"
  - Lösung Objekt-basiert (C)
  - Lösung Objekt-orientiert (C++)
- Low-Level Treiber in C und C++
- Zusammenfassung



Eine Vielzahl von Embedded Systemen wurde in der Vergangenheit – und wird sicher auch noch in Zukunft – in C programmiert.
Hier kommt als Programmier-Paradigma die „Objekt-basierte Programmierung“ ins Spiel. Und dann stellt sich als nächstes die Frage ob nicht gleich mit C++ „Objekt-orientiert“ programmiert werden soll.

Die Programmiersprache C

C wurde 1969–1973 von Dennis Ritchie in den Bell Laboratories für die Programmierung des Betriebssystems UNIX entwickelt.

C zählt zu den **prozeduralen Programmiersprachen** und wurde mehrfach standardisiert (C89/C90, C95, C99, C11).

Für die Anwendung im Embedded Bereich für Mikrocontroller gibt es typisch Spracherweiterungen (Dialoge).

➪ C-Programme, die sehr hardware-nahe (Mikrocontroller-bezogene) Programmierung enthalten, sind daher in der Regel nicht so einfach portierbar wie C-Systeme für PC/Server-Applikationen.

Die Programmiersprache C++

C++ wurde ab 1979 von Bjarne Stroustrup bei AT&T als objekt-orientierte Erweiterung der Programmiersprache C ("C mit Klassen") entwickelt.

C++ zählt zu den **objekt-orientierten Programmiersprachen**, kann aber aufgrund der Kompatibilität zur Programmiersprache C auch für rein prozedurale Programmierung verwendet werden.

C++ wurde mehrfach standardisiert (C++98, C++03, C++11, C++14).

Für die Anwendung im Embedded Bereich für Mikrocontroller gibt es wie bei C Spracherweiterungen.

Agenda

- Programmier-Paradigmen
- Programmiersprachen C und C++
- **Aufgabenstellung "Zähler"**
  - **Lösung Objekt-basiert (C)**
  - **Lösung Objekt-orientiert (C++)**
- Low-Level Treiber in C und C++
- Zusammenfassung

Zähler

Zähler werden in verschiedensten Applikationen benötigt und können in vielen unterschiedlichen Formen auftreten:

- Grenzwertzähler (z.B. Uhr mit Sekunden, Minuten, . . .)
- Frei laufender Aufwärtszähler (z.B. Mengenmessung)
- Auf-/Abwärtszähler (z.B. freie Parkplätze im Parkhaus)
- . . .

Aufgabenstellung Zähler

Das System benötigt einen Zähler, der beginnend mit dem Start des Systems die Laufzeit in Sekunden messen soll.

- Dafür wird ein Aufwärtszähler benötigt.

Objekt-basiert oder objekt-orientiert? – C Beispielcode – Headerfile

```
/*
 * counter.h
*/
#ifndef COUNTER_H_
#define COUNTER_H_
#include <stdint.h>

typedef struct counter counter_t;

// Zaehler fuer aufwaerts zaehlen
struct counter
{
   int32_t mCountValue;
};

// Prototypen der Zugriffsfunktionen
void count(counter_t* const pCounter, int32_t stepValue);
void setCountValue(counter_t* const pCounter, int32_t countValue);
int32_t getCountValue(const counter_t* const pCounter);

#endif /* COUNTER_H_ */
```

Definition des Datentypnamens

Definition des Datentyps

Objekt-basiert oder objekt-orientiert? – C Beispielcode – Implementierungsfile

```
/*
 * counter.c
*/
#include "counter.h"

/*  *  */
void count(counter_t* const pCounter, int32_t stepValue)
{
   pCounter->mCountValue += stepValue;
}

/*  *  */
void setCountValue(counter_t* const pCounter, int32_t countValue)
{
   pCounter->mCountValue = countValue;
}

/*  *  */
int32_t getCountValue(const counter_t* const pCounter)
{
   return pCounter->mCountValue;
}
```

Aufwärts zählen um Schrittweite

Startwert setzen

Zählwert als Rückgabewert

Objekt-basiert oder objekt-orientiert? – C Beispielcode – Applikationsfile

```
/*
 * counterMain.c
*/
#include "counter.h"

counter_t runtimeCounter;

int main(void)
{
   // Initialize the microcontroller system

   setCountValue(&runtimeCounter, 0);



   while(1)
   {
      // . . .
   }
}
```

Globale Variable vom Typ `counter_t`

Initialisierung der globalen Variable
- Die Adresse der Variable `runtimeCounter` muss als Übergabeparameter an die Funktion übergeben werden.

Das Zählen der globalen Variable `runtimeCounter` findet in einer Interrupt-Callbackfunktion statt

Objekt-basiert oder objekt-orientiert? – C++ Beispielcode – Headerfile (1..)

```
/*   * counter.h
*/
#ifndef COUNTER_H_
#define COUNTER_H_
#include <stdint.h>

// Zaehler fuer aufwaerts zaehlen
class Counter
{
public:
   Counter(int32_t countInitVal)
   {  this->mCountValue = countInitVal;
   }
   void count(int32_t stepValue)
   {  this->mCountValue += stepValue;
   }
   void setCountValue(int32_t countValue)
   {  this->mCountValue = countValue;
   }
   int32_t getCountValue() const   { return this->mCountValue; }

private:
   int32_t mCountValue;
};
#endif /* COUNTER_H_ */
```

Definition des Abstrakten Datentyps: der 1. Buchstabe des Klassennamens wird typisch gross geschrieben

Methoden der Klasse sind typisch **`public`**

Datenelemente der Klasse sind typisch **`private`** (gekapselt)

Datenelement der Klasse

Objekt-basiert oder objekt-orientiert? – C++ Beispielcode – Headerfile (..2)

```
/*   * counter.h
*/
#ifndef COUNTER_H_
#define COUNTER_H_
#include <stdint.h>

// Zaehler fuer aufwaerts zaehlen
class Counter
{
public:
   Counter(int32_t countInitVal)
   {  this->mCountValue = countInitVal;
   }
   void count(int32_t stepValue)
   {  this->mCountValue += stepValue;
   }
   void setCountValue(int32_t countValue)
   {  this->mCountValue = countValue;
   }
   int32_t getCount
                                    re
private:
   int32_t mCountValue;
};
#endif /* COUNTER_H_ */
```

Konstruktor der Klasse: Initialisiert die Datenelemente und wird automatisch vom Compiler aufgerufen

Methoden die direkt in der Klasse implementiert werden, sind **implizit inline**

Objektzeiger `this` wird automatisch vom Compiler als 1. Übergabe-parameter eingesetzt

`this` muss nicht geschrieben werden

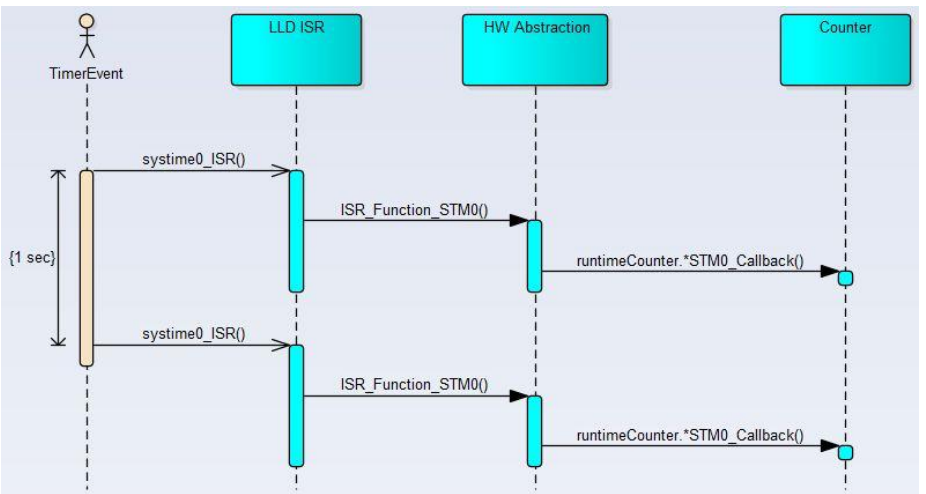
Objekt-basiert oder objekt-orientiert? – C++ Beispielcode – Applikationsfile

```
/*
 * counterMain.c
*/
#include "counter.h"

Counter runtimeCounter(0);

int main(void)
{

   // Initialize the microcontroller system

   while(1)
   {
      // . . .
   }
}
```

Objekt vom Typ Counter:
Der Konstruktor wird beim Erzeugen des Objekts automatisch vom Compiler aufgerufen

Das Zählen des Objekts `runtimeCounter` findet in einer Interrupt-Callbackfunktion statt

Objekt-basiert oder objekt-orientiert? – C++ Beispielcode – Sequenzdiagramm



Watch View1

Name	Value
runtimeCounter	0xD0000018
mCountValue	127

Objekt-basiert oder objekt-orientiert? – Aufgabenstellung Zähler – Erweiterung

Aufgabenstellung Zähler – Erweiterung

Das System benötigt einen Zähler, der beginnend mit dem Start des Systems die Laufzeit in Sekunden messen soll.

- Dafür wird ein Aufwärtszähler benötigt.

Das System benötigt einen Zähler, der abhängig von einer voreingestellten Zählrichtung zählen soll.

- Dafür wird ein Auf- oder Abwärtszähler benötigt.

Objekt-basiert oder objekt-orientiert? – C Beispielcode – Headerfile

```
/*   * counter.h
*/
#ifndef COUNTER_H_
#define COUNTER_H_
#include <stdint.h>

typedef struct counter counter_t;
typedef enum counterDIR counterDir_t;

enum counterDIR  { UP = 0, DOWN = 1, };

// Zaehler fuer auf- oder abwaerts zaehlen
struct counter
{
   int32_t mCountValue;
   counterDir_t mCountDir;
};

// Prototypen der Zugriffsfunktionen
void count(counter_t* const pCounter, int32_t stepValue);
void initCounter(counter_t* const pCounter, int32_t countValue
                 , counterDir_t countDir);
void setCountValue(counter_t* const pCounter, int32_t countValue);
int32_t getCountValue(const counter_t* const pCounter);
#endif /* COUNTER_H_ */
```

Definition des Datentypnamens

Aufzählungstyp für Zählrichtungen

Vorhandener Datentyp wird erweitert

Neues Element für Zählrichtungseinstellung

Zugriffsfunktionen müssen erweitert werden

Objekt-basiert oder objekt-orientiert? – C Beispielcode – Implementierungsfile

```
/*  * counter.c
*/
#include "counter.h"

/*  *  */
void count(counter_t* const pCounter, int32_t stepValue)
{
   if(pCounter->mCountDir == UP)
   {
      pCounter->mCountValue += stepValue;
   }
   else
   {
      pCounter->mCountValue -= stepValue;
   }
}
/*  *  */
void initCounter(counter_t* const pCounter, int32_t countValue
                 , counterDir_t countDir)
{
   pCounter->mCountValue = countValue;
   pCounter->mCountDir   = countDir;
}

// . . .
```

Die Funktion `count()` muss abhängig von der Zählrichtung addieren oder subtrahieren.

Die Funktion `initCounter()` muss auch die Zählrichtung initialisieren.

Objekt-basiert oder objekt-orientiert? – C Beispielcode – Applikationsfile

```
/*  * counterMain.c
*/
#include "counter.h"

counter_t runtimeCounter;

#define WAIT_TIME 0x3FFFFF

int main(void)
{
   static counter_t downCounter;

   // Initailze the microcontroller system

   initCounter(&runtimeCounter, 0, UP);
   initCounter(&downCounter, WAIT_TIME, DOWN);

   while(1)
   {  count(&downCounter, 1);
      if(getCountValue(&downCounter) == 0)
      {
         toggle_led(P33_BASE, 0);
         setCountValue(&downCounter, WAIT_TIME);
      }
   }
}
```

Globale Variable vom Typ `counter_t`

Alle Variablen vom Typ `counter_t` können auf- oder abwärts zählen

lokale Variable vom Typ `counter_t`

Initialisierung als Up-Counter

Initialisierung als Down-Counter

Der Abwärtszähler wird für die Warteschleife genutzt.

Objekt-basiert oder objekt-orientiert? – C++ Beispielcode – Headerfile

```
/*   * counter.h
*/
#ifndef COUNTER_H_
#define COUNTER_H_
#include <stdint.h>

// Zaehler fuer aufwaerts zaehlen
class Counter
{
public:
   Counter(int32_t countInitVal)
   {  this->mCountValue = countInitVal;
   }
   void count(int32_t stepValue)
   {  this->mCountValue += stepValue;
   }
   void setCountValue(int32_t countValue)
   {  this->mCountValue = countValue;
   }
   int32_t getCountValue() const   { return this->mCountValue; }

private:
   int32_t mCountValue;
};
#endif /* COUNTER_H_ */
```

Die bereits programmierte und getestete Klasse **`Counter`** bleibt unverändert!

Objekt-basiert oder objekt-orientiert? – C++ Beispielcode – Headerfile

```
/*    * counterUpDown.h
*/
#ifndef COUNTERUPDOWN_H_
#define COUNTERUPDOWN_H_
#include "counter.h"

enum counterDIR  { UP = 0, DOWN = 1, };

// Zaehler fuer auf- oder abwaerts zaehlen
class CounterUpDown : public Counter
{
public:
   CounterUpDown(int32_t countInitVal, counterDir countDir)
      : Counter(countInitVal), this->mCountDir = countDir;
   }
   void count(int32_t stepValue)
   {
      if(this->mCountDir == UP)  { this->mCountValue += stepValue; }
      else                       { this->mCountValue -= stepValue; }
   }

private:
   const counterDir mCountDir;
};
#endif /* COUNTERUPDOWN_H_ */
```

Die Definition der Basisklasse Counter muss dem Compiler zur Verfügung stehen.

Aufzählungstyp für Zählrichtungen

Die Klasse `CounterUpDown` erbt von der Klasse `Counter`

Der Konstruktor leitet den Initialisierungswert für das geerbte Datenelement an den Konstruktor der Basisklasse weiter

Geerbte Methoden der Basisklasse können überschrieben werden

Die Zählrichtung soll read-only sein. Beim erzeugen des Objekts kann sie vom Konstruktor der Klasse initialisiert werden.

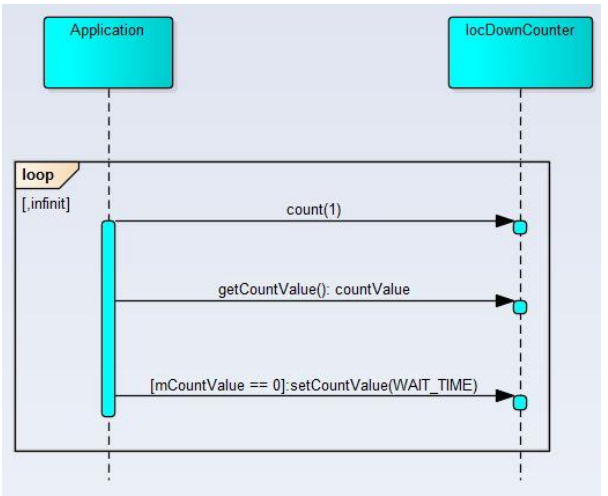
Objekt-basiert oder objekt-orientiert? – C++ Beispielcode – Applikationsfile

```
/*
 * counterMain.c
*/
#include "counter.h"
#include "counterUpDown.h"

#define WAIT_TIME 0x3FFFFF

Counter runtimeCounter(0);

int main(void)
{
   static CounterUpDown locDownCounter(WAIT_TIME, DOWN);

   // Initialize the microcontroller system

   while(1)
   {
      locDownCounter.count(1);
      if(locDownCounter.getCountValue() == 0)
      {
         toggleLED(0);
         locDownCounter.setCountValue(WAIT_TIME);
      }
   }
}
```

lokales statisches Objekt vom Typ `CounterUpDOWN` Der Konstruktor initialisiert das read-only Datenelement für die Zählrichtung abwärts

Der Abwärtszähler wird für die Warteschleife genutzt.

Objekt-basiert oder objekt-orientiert? – C++ Beispielcode – Sequenzdiagramm



Watch View1

Name	Value
{} locDownCounter	0xD000034C
mCountValue	0x3C3F07
mCountDir	DOWN

MICRO CONSULT

Objekt-basiert oder objekt-orientiert? – Agenda

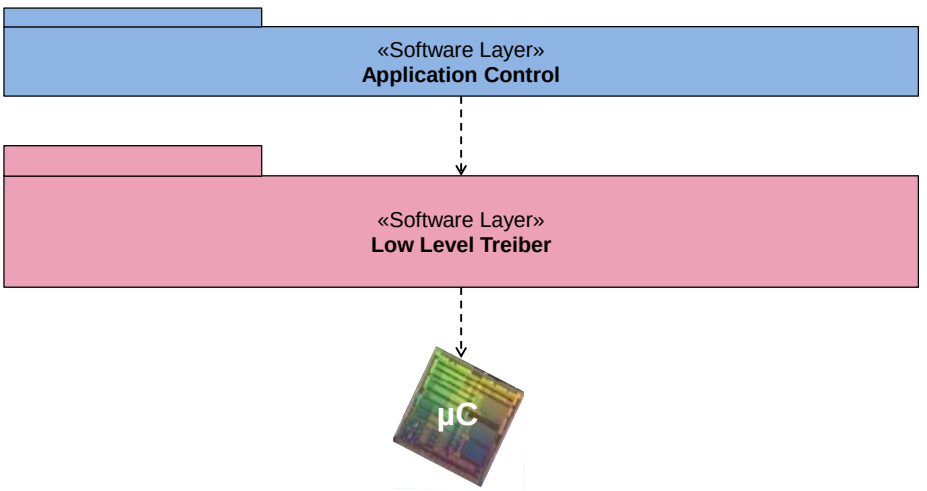
Agenda

- Programmier-Paradigmen
- Programmiersprachen C und C++
- Aufgabenstellung "Zähler"
  - Lösung Objekt-basiert (C)
  - Lösung Objekt-orientiert (C++)
- **Low-Level Treiber in C und C++**
- Zusammenfassung

MICRO CONSULT

Objekt-basiert oder objekt-orientiert? – Software Schichtenmodell

2-Schichten-Modell



MICRO CONSULT

Objekt-basiert oder objekt-orientiert? – Mikrocontroller

Beispiel Infineon TC299 - BGA516 Gehäuse

- 22 Port Module
  mit insgesamt 276 programmierbaren Ein-/Ausgangspins
- **Ein Satz Steuer-/Status-/Arbeitsregister**
  pro Port Modul

ACCEN0
ACCEN1
res
OMCR
OMSR
OMCR12
OMCR8
OMCR4
OMCR0
OMSR12
OMSR8
OMSR4
OMSR0
res
PDISC
res
ESR
res
PDR1
PDR0
res
IN
res
IOCR12
IOCR8
IOCR4
IOCR0
res
ID
OMR
OUT

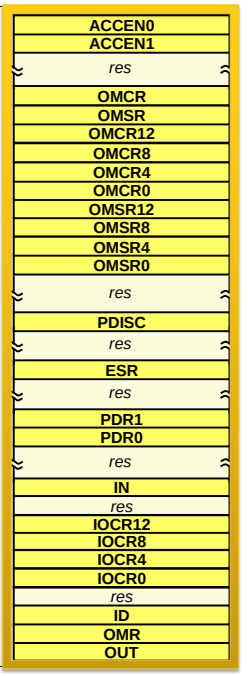
Port Modul Basisadresse →

MICRO CONSULT

Objekt-basiert oder objekt-orientiert? – Treiber – µC Modulabbild in C

```
typedef struct port
{
    volatile unsigned int OUT;
    volatile unsigned int OMR;
    volatile unsigned int ID;
    volatile unsigned int reserved0[1];
    volatile unsigned int IOCR0;
    volatile unsigned int IOCR4;
    volatile unsigned int IOCR8;
    volatile unsigned int IOCR12;
    volatile unsigned int reserved1[1];
    volatile unsigned int IN;
    volatile unsigned int reserved2[6];
    volatile unsigned int PDR0;
    volatile unsigned int PDR1;
    volatile unsigned int reserved3[2];
    volatile unsigned int ESR;
    volatile unsigned int reserved4[3];
    volatile unsigned int PDISC;
    volatile unsigned int reserved5[3];
    volatile unsigned int OMSR0;
    volatile unsigned int OMSR4;
    volatile unsigned int OMSR8;
    volatile unsigned int OMSR12;
    volatile unsigned int OMCR0;
    volatile unsigned int OMCR4;
    volatile unsigned int OMCR8;
    volatile unsigned int OMCR12;
    volatile unsigned int OMSR;
    volatile unsigned int OMCR;
    volatile unsigned int reserved6[24];
    volatile unsigned int ACCEN1;
    volatile unsigned int ACCEN0;
} PORT;
```

Definition des Registerblocks für ein Port Modul



Basisadresse →

MICRO CONSULT

Objekt-basiert oder objekt-orientiert? – Treiber in C programmieren

Prototypen der Zugriffsfunktionen

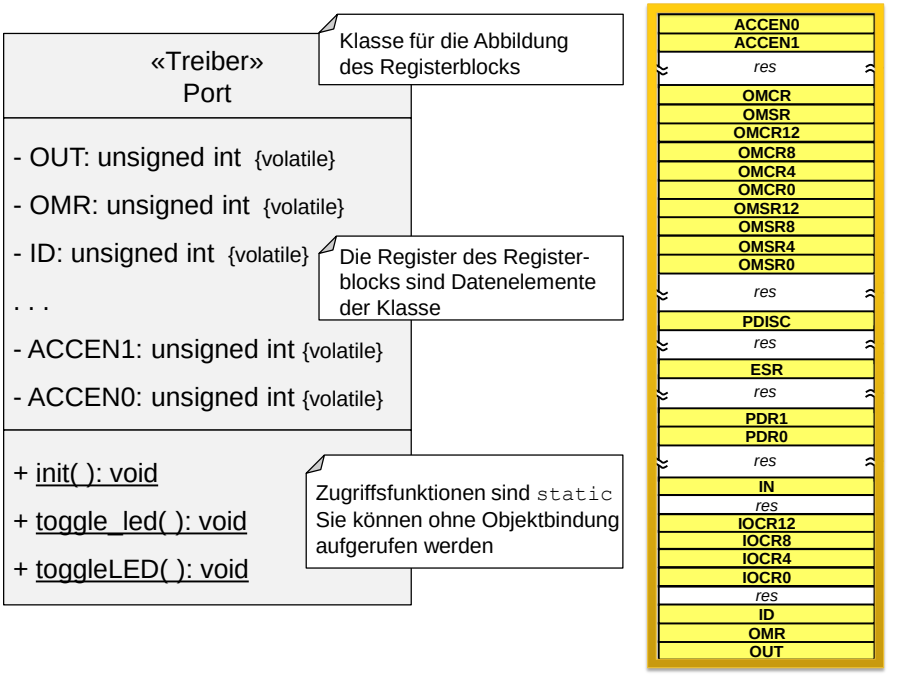
```
void port_init(PORT* const pBase
               ,const PInitStruct_t* const pInitStruct);
void toggle_led(PORT* const pBase, uint32_t Index);
void toggleLED(uint32_t Index);
```

```
#define P00_BASE((PORT*)(0xF003A000))
#define P01_BASE((PORT*)(0xF003A100))
#define P02_BASE((PORT*)(0xF003A200))
#define P10_BASE((PORT*)(0xF003B000))
#define P11_BASE((PORT*)(0xF003B100))

// . . .

#define P33_BASE((PORT*)(0xF003D300))
#define P34_BASE((PORT*)(0xF003D400))
#define P40_BASE((PORT*)(0xF003E000))
```

Definition der Basisadressen der Port Module

MICRO CONSULT

Objekt-basiert oder objekt-orientiert? – Treiber – Modulabbild in C++



Objekt-basiert oder objekt-orientiert? – Treiber in C++ programmieren

```
class Port
{
public:
   static void init(Port* const pBase
                   ,const pInitStruct* const pInitStruct);
   static void toggle_led(Port* const pBase
                          , uint32_t Index);
   static void toggleLED(uint32_t Index);
private:
   volatile unsigned int OUT;
   volatile unsigned int OMR;
   volatile unsigned int ID;
   // . . .
   volatile unsigned int ACCEN1;
   volatile unsigned int ACCEN0;
};

#define P00_BASE((Port*)(0xF003A000))
#define P01_BASE((Port*)(0xF003A100))
// . . .
#define P40_BASE((Port*)(0xF003E000))
```

Definition der Klasse

Prototypen der `static` Zugriffsfunktionen

Definition des Registerblocks für ein Port Modul als private Datenelemente der Klasse

Definition der Basisadressen der Port Module

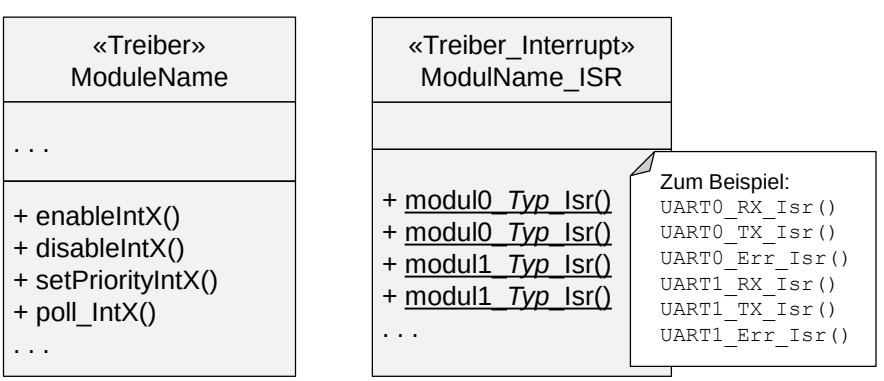
Objekt-basiert oder objekt-orientiert? – Treiber – Interrupts

- Die Abbildung von Interrupts ist stark von der individuellen Hardware des Interrupt-Controllers abhängig.
- Dem Device zugehörige Interrupt-Funktionalitäten werden im Treiber umgesetzt.
- Der Interrupt Controller wird selbst über einen Treiber abgebildet.
- Interrupt-Bearbeitung erfolgt über ISRs bzw. Tasks und Callback-Mechanismen.
- ISRs sind in C/C++ NICHT standardisiert.
- ISRs können in Klassen statische Operationen sein (unabhängig von der Klasseninstanz).
- Durch die Einbindung von ISRs darf der objektorientierte Ansatz nicht missachtet werden.

Objekt-basiert oder objekt-orientiert? – Treiber – Interrupts

Annahmen:

- Jedes Peripheriemodul enthält Konfigurationsmöglichkeiten für Interrupts.
- Jede Einheit benötigt ihre eigenen Interrupt Service Routinen.



Agenda

- Programmier-Paradigmen
- Programmiersprachen C und C++
- Aufgabenstellung "Zähler"
  - Lösung Objekt-basiert (C)
  - Lösung Objekt-orientiert (C++)
- Low-Level Treiber in C und C++
- **Zusammenfassung**

- Objekt-orientierte Programmierung mit einer geeigneten Programmiersprache vereinfacht und erleichtert die Programmierung – auch in Embedded Systemen – und auch die Low-Level-Treiber Programmierung.
- Die Effizienz in Form von Speicherbedarf und/oder Laufzeit leidet nicht darunter, wenn vorher schon objekt-basiert, also mit Strukturen und Zugriffsfunktionen, gearbeitet wurde und konsequent mit `inline` Methoden gearbeitet wird.

Objekt-basiert oder objekt-orientiert? – Kontakt

Renate Schultes
MicroConsult GmbH
Trainer & Coach
Mikrocontroller Hardware & Software

r.schultes@microconsult.de
www.microconsult.de

Im Internet sind die C und C++ Beispiele als Download unter folgender Adresse verfügbar:

http://download.microconsult.net/rs/ese-2015/c_cpp_beispielcode.zip

Bitte beachten Sie, dass der komplette Pfad inklusive Dateinamen angegeben wird.
Username oder Passwort sind nicht erforderlich.

MicroConsult – www.microconsult.de